New Technical Notes

Macintosh

Developer Support

MacsBug Q&As

Platforms & Tools

Revised by: Developer Support Center May 1993
Written by: Developer Support Center October 1990

This Technical Note contains a collection of Q&As relating to a specific topic—questions you've sent the Developer Support Center (DSC) along with answers from the DSC engineers. While DSC engineers have checked the Q&A content for accuracy, the Q&A Technical Notes don't have the editing and organization of other Technical Notes. The Q&A function is to get new technical information and updates to you quickly, saving the polish for when the information migrates into reference manuals.

Q&As are now included with Technical Notes to make access to technical updates easier for you. If you have comments or suggestions about Q&A content or distribution, please let us know by sending an AppleLink to DEVFEEDBACK. Apple Partners may send technical questions about Q&A content to DEVSUPPORT for resolution.

New Q&As this month:
Determining whether MacsBug debugger is installed

Determining whether MacsBug debugger is installed
Date Written: 9/9/92
Last reviewed: 3/1/93

I'm attempting to determine whether a debugger is installed, and if so, to find a THz pointer to its heap zone. Is this possible?

—

The MacsBug debugger is loaded into high memory above the value found in the global variable BufPtr ($10C). Since it's loaded into the memory that's not managed by the Memory Manager, it's not in a heap. The global variable MacJmp ($120) points to the debugger's entry point.

There's also a flags byte in low memory that contains the following information:

```
Bit 7  Set if debugger is running.
Bit 6  Set if debugger can handle system errors.
Bit 5  Set if debugger is installed.
```

```
Bit 4  Set if debugger can support the Discipline utility.
```

The flags byte may be in one of two places: the high-order byte of the long word at MacJmp, or the address $BFF. When MacsBug is loaded, it examines the value at address $BFF. If the

value at $BFF is $FF, the system is using the 24-bit Memory Manager and the flags byte will be the high-order byte of the long word at MacJmp. If the value at $BFF isn't $FF, the system is using the 32-bit Memory Manager and the flags byte will be at address $BFF.

For information on debuggers other than MacsBug, you'll need to contact the publishers of those products.

Macintosh IIfx MacsBug requirements

Date Written: 7/6/90
Last reviewed: 8/30/91

What version of MacsBug is required for a Macintosh IIfx?

—

The Macintosh Technical Note "IIfx: The Inside Story" gives you Macintosh IIfx hardware and software details, including what version of MacsBug you need to use. You must have MacsBug 6.2 or later, as earlier versions are incompatible with the new hardware. You can get the latest MacsBug from APDA, the current *Developer CD Series* disc, and on AppleLink (in the Tools folder of Macintosh DTS). You can download this Tech Note from AppleLink if you don't have a copy.

Macintosh 'dcmd' resource format

Date Written: 7/25/90
Last reviewed: 10/1/91

What is the format of a 'dcmd' resource?

—

The following information is taken from the MacsBug Reference manual, available from APDA:

A 'dcmd' resource begins with a 4-byte segment header, followed immediately by the program code. Since a 'dcmd' is limited to single segment, the segment header is used to specify a 'dcmd' version number and the amount of global variable space MacsBug needs to allocate for the 'dcmd' (that is, the size of its "A5 world")—word of version followed by word of size.

MacsBug heap check/scramble/purge dcmd

Date Written: 1/18/91
Last reviewed: 2/14/91

Is there a way to get MacsBug to do a heap purge in the same way that TMON does? I would like to be able to run my programs with a heap check/scramble/purge performed automatically after each trap. Would it help to write a dcmd?

—

Both the Product Manager and Engineer on the product said MacsBug can't do a heap purge,

so I tried the alternative you suggested, writing a dcmd.

This solution seems to work out well. I patch NewPtr, NewHandle, SetPtrSize, SetHandleSize, ReallocHandle, and MoveHHi. Whenever one of these is called, and the user has turned on heap purge (with “hp” in MacsBug), all purgeable handles are dumped by calling EmptyHandle.

I’ve included the MPW C source code below for you to play around with. I’ve also suggested to the Product Manager that heap purge be something we put into MacsBug in the future.

```
-----------
File: HP.c
-----------

/*  HP.c
 This is the Heap Purge dcmd.
```

The following MPW commands will build the dcmd and copy it to the Debugger Prefs file in the System folder. The 'dcmd's name in MacsBug will be the name of the file built by the Linker. You must first copy dcmd.h, dcmdGlue.a.o and DRunTime.o from the C Samples folder into this folder.

```
 Asm hp.a
 C -r -b hp.c #-sym on,novars,notypes -mbg off # for UltraSlimFast
 Link dcmdGlue.a.o hp.c.o hp.a.o put.c.o DRuntime.o "{Libraries}"Interface.o
   -o hp
 BuildDcmd hp 1235
 #Echo 'include "hp";'  |  Rez -a -o "{SystemFolder}TMON Folder:DCMD Holder"
 Echo 'include "hp";'   |  Rez -a -o "{SystemFolder}Debugger Prefs"

 DumpObj hp.c.o > hp.dumpObj
 UltraSlimFast hp.dumpObj >hp.UltraSlimFast

*/

#include <Types.h>
#include <Memory.h>
#include <OSUtils.h>
#include <Files.h>
#include <Menus.h>
#include <Traps.h>

#include "dcmd.h"
#include "put.h"

extern pascal void NewNewPtr();
extern pascal void NewNewHandle();
extern pascal void NewReallocHandle();
extern pascal void NewSetPtrSize();
extern pascal void NewSetHandleSize();
extern pascal void NewMoveHHi();
extern pascal void SaveOldTrapAddress (long address, short addressKind);
extern pascal void SaveMyA5();
extern pascal long GetMyA5();

#define kOldNewPtr           0
#define kOldNewHandle        1
#define kOldReallocHandle    2
#define kOldSetPtrSize       3
#define kOldSetHandleSize    4
#define kOldMoveHHi          5
```

```
#define sysZoneMask          0x0400

Str255        pDumpString;
char          ch;
Boolean       purgingOn;

pascal void PurgeIt(long blockAddress, long blockLength, long addrOfMasterPtr,
                    short blockType, Boolean locked, Boolean purgeable,
                    Boolean resource)
{
#pragma unused (blockAddress, blockLength, addrOfMasterPtr, resource)

   if ((blockType == relocatableBlock) && (!locked) && (purgeable)) {
       EmptyHandle((Handle)addrOfMasterPtr);
   }
}

Boolean    IsSysZone(unsigned short TrapWord)
{
   return( (TrapWord & sysZoneMask) != 0);
}

pascal void PurgeAllBlocks(unsigned short TrapWord)
{
   THz     oldZone;
   long    oldA5;

   oldA5 = SetA5(GetMyA5());
   if (purgingOn) {
       if (IsSysZone(TrapWord)) {
           dcmdSwapWorlds();
           oldZone = GetZone();
           SetZone(SystemZone());
           dcmdSwapWorlds();
       }

       dcmdForAllHeapBlocks(PurgeIt);

       if (IsSysZone(TrapWord)) {
           dcmdSwapWorlds();
           SetZone(oldZone);
           dcmdSwapWorlds();
       }
   }
   (void) SetA5(oldA5);
}

TrapType GetTrapType(short theTrap)
{
   // OS traps start with A0, Tool with A8 or AA.
   return((theTrap & 0x0800) ? ToolTrap : OSTrap);
}

void PatchTrap(short trapNumber, short saveOffset, long newAddress)
{
   // Use NGetTrapAddress since it is always safer on current machines. Take
   // the result it gives me, and save it off in asm land, for future
   // reference. Then, move in the new address of the routine, my asm glue.

   SaveOldTrapAddress(NGetTrapAddress(trapNumber, GetTrapType(trapNumber)),
                      saveOffset);
   NSetTrapAddress(newAddress, trapNumber, OSTrap);
}
```

```
void InstallPatches()
{
   // Patch the traps… These are being patched in the world, not in the
   // debugger world. Switch over to the real world, in case the debugger
   // does world swaps. TMon Pro.

   dcmdSwapWorlds();

   PatchTrap(_NewPtr, kOldNewPtr, (long) NewNewPtr);
   PatchTrap(_NewHandle, kOldNewHandle, (long) NewNewHandle);
   PatchTrap(_ReallocHandle, kOldReallocHandle, (long) NewReallocHandle);
   PatchTrap(_SetPtrSize, kOldSetPtrSize, (long) NewSetPtrSize);
   PatchTrap(_SetHandleSize, kOldSetHandleSize, (long) NewSetHandleSize);
   PatchTrap(_MoveHHi, kOldMoveHHi, (long) NewMoveHHi);

   // Switch back to debugger world.
   dcmdSwapWorlds();
}

pascal void CommandEntry(dcmdBlock* paramPtr)
{
   switch (paramPtr->request)
       {
       case dcmdInit:
           SaveMyA5();
           purgingOn = false;
           InstallPatches();
           break;

       case dcmdHelp:
           dcmdDrawLine("\php");
           dcmdDrawLine("\p   Toggle Heap Purge on and off. When on, heap
purge purges purgable blocks");
           dcmdDrawLine("\p   on NewPtr, NewHandle, ReallocHandle,
SetPtrSize, SetHandleSize, and MoveHHi");
           dcmdDrawLine("\p   calls.");
           break;

       case dcmdDoIt:
           dcmdDrawLine("\pHeap purge is ");
           if (purgingOn)
               dcmdDrawString("\poff.");
           else
               dcmdDrawString("\pon.");
           purgingOn = !purgingOn;
           break;

       default:
           PutPStr("\pUnknown request ");
           PutUDec(paramPtr->request);
           PutLine();
           break;
       }
} // CommandEntry

-----------
File: HP.a
-----------
           include        'SysErr.a'
           include        'SysEqu.a'
           include        'Traps.a'
```

```
allRegs         REG             D0-D7/A0-A5

            proc

            import          PurgeAllBlocks

; ____________________________________________________________________________
;

            export      SaveMyA5, GetMyA5

pMyA5         dc.l      0

SaveMyA5
            lea          pMyA5,A0
            move.l      A5,(A0)
            rts

GetMyA5
            move.l      pMyA5,4(SP)
            rts


; ____________________________________________________________________________
;
; Storage for the old patch addresses, used to call through once the patch
; code executes. These are essentially globals, used by the asm code. They are
; specifically not exported, so that the Pascal code cannot access them
; directly. There are a number of interface routines I set up so that Pascal
; can get and set them, but has to go through this file. You know, sort of
; object like.
;
pFirstSavedTrap             equ         *
pOldNewPtr                  dc.l        0
pOldNewHandle               dc.l        0
pOldReallocHandle           dc.l        0
pOldSetPtrSize              dc.l        0
pOldSetHandleSize           dc.l        0
pOldMoveHHi                 dc.l        0

; ____________________________________________________________________________
; When I'm am setting up the world, I call NGetTrapAddress to get the old
; version of the traps. I need to save that dude off so I can get back there
; when needed. This routine is a handy interface to the high-level world,
; isolating this asm junk from the code. All these routines are the same, just
; a different variable being affected. This hunk uses the PC-Relative
; addressing mode in order to get the address of the variable being set. This
; allows the code to function without any explicit global space, since the
; code acts like globals here.
;
; The interface is:
;
;    PROCEDURE  SaveOldTrapAddress (address: LongInt; addressKind: Integer);
;

            export          SaveOldTrapAddress

SaveOldTrapAddress
            MOVE.L          (SP)+,A0                ; get the return address
            move.w          (SP)+,D0
            asl.w           #2,D0
            LEA             pFirstSavedTrap,A1      ; the variable to be setting
```

```
            MOVE.L      (SP)+,(A1,D0.w)   ; save it, pulling parameter too
            JMP         (A0)              ; it's saved, return to high-level

            export      NewNewPtr, NewNewHandle, NewReallocHandle
            export      NewSetPtrSize, NewSetHandleSize, NewMoveHHi

NewNewPtr
            move.l      pOldNewPtr,-(SP)
            bra.s       Common

NewNewHandle
            move.l      pOldNewHandle,-(SP)
            bra.s       Common

NewReallocHandle
            move.l      pOldReallocHandle,-(SP)
            bra.s       Common

NewSetPtrSize
            move.l      pOldSetPtrSize,-(SP)
            bra.s       Common

NewSetHandleSize
            move.l      pOldSetHandleSize,-(SP)
            bra.s       Common

NewMoveHHi
            move.l      pOldMoveHHi,-(SP)
;             bra.s      Common

Common
            movem.l     allRegs,-(SP)
            move.w      D1,-(SP)
            bsr         PurgeAllBlocks
            movem.l     (SP)+,allRegs
            rts

            ENDP
            END
```

MacsBug and TMON now support longer compiler symbol names

Date Written: 5/3/89
Last reviewed: 12/17/90

Why don’t the MPW compilers generate debugger symbols which can be recognized by TMON or MacsBug?

—

The “-mbg ch8” option to the MPW 3.0 compilers cause the compilers to generate old style 8-character symbol names used by older versions of MacsBug and TMON. The default longer symbol names are necessary for use with object-oriented languages such as Object Pascal and C++, which generate very long symbol names. These names are supported by newer versions of MacsBug and TMON.

MacsBug Heap Zone “!” flag

Date Written: 4/10/91

Last reviewed: 10/9/91

What does the "!" mean when I use the MacsBug Heap Zone (HZ) command? It appears in front of one of the zone names listed, or just after the address if the zone doesn't have a name.

—

MacsBug's HZ command does a quick-and-dirty heap check, and if it thinks something is wrong with a heap, it puts the exclamation point after the address range of the heap. If you select the heap flagged with a "!" with the Heap Exchange (HX) command and then use the regular Heap Check (HC) command, MacsBug tells you what it thinks is wrong with that heap.

Trick for accessing MacsBug TargetZone variable
Date Written: 8/12/91
Last reviewed: 9/17/91

How can I get the target heap for my MacsBug dcmd? My command currently operates on whichever heap is in TheZone. It seems better to be operating on the target heap as that is what other currently available heap commands do.

—

In recent versions of MacsBug, there's a predefined variable that contains the target heap zone; as you might guess, it's "TargetZone." Currently there's no call-back to allow your dcmd to access this variable, but here's a workaround: Write your dcmd as though the first parameter is always TargetZone, and the second parameter is an optional override for the first. Then, always call your dcmd through a macro; name the dcmd "htx," and create a macro "ht" that expands to "htx TargetZone." This will give your dcmd access to the TargetZone variable.

X-Ref:
MacsBug User's Guide and Reference—for MacsBug 6.2, Addison-Wesley

Debugger Prefs HFSDispatch macros fixed on MacsBug 6.2.2
Date Written: 8/29/91
Last reviewed: 9/16/91

Why don't the "HFSDispatch" macros that are included in the MacsBug 6.2 Debugger Prefs file work? When we try any of those macros, we get the error message "unrecognized symbol 'HFSDispatch'" from MacsBug.

—

You must be using MacsBug 6.2.1. During the build of that version of MacsBug, one file

was left out of the build script by mistake and it had some predefined constants that are needed by some macros. This has been fixed with MacsBug 6.2.2, available on the latest *Developer CD Series* disc.

Byte-wide accesses for db, dw, and dl commands
Date Written: 10/3/91
Last reviewed: 10/15/91

Macsbug uses byte accesses when I issue the dw command in slot space to display a word.

—

MacsBug always makes byte-wide accesses to fetch data for the db, dw, or dl commands. This is to avoid errors that might be caused if you supplied an odd address on a 68000-based Macintosh.

To accomplish what you need to do, write a tiny 'dcmd' which explicity fetches and displays a word. This should only need to be a couple of lines long. How to write a 'dcmd' is documented in the files accompanying MacsBug on the *Developer CD Series* discs.

How to tell whether a Macintosh debugger is installed
Date Written: 11/12/91
Last reviewed: 12/11/91

What is a robust way to tell if a debugger is installed? According to the MacsBug Reference, a low-memory global, MacJmp, contains the address of any debugger, but the manual doesn’t say (1) whether this address is NIL or garbage if no debugger is installed, and (2) what the actual address of MacJmp is.

—

To determine if a debugger is installed, you do want to examine the low-memory global MacJmp vector; it is at address $120 and is 4 bytes long. When a debugger is not installed, the value there is 0. When a debugger is installed, the value there is non-zero. This is true for MacsBug and TMON, and probably for The Debugger. This is the standard way to determine if a debugger is installed; third-party debuggers should set this up properly.

X-Ref: “How MacsBug Installs Itself,” *MacsBug Reference,* page 412.

MacsBug and DebugStr documentation
Date Written: 3/12/91
Last reviewed: 7/25/91

Where can I find documentation on the Debugger and DebugStr calls? Also, I'm fairly certain I've seen a MacsBug manual somewhere, but I've checked the Dev CD (Vol. VI: Gorillas) and find only a beta copy, and delta documentation. I've checked the TN stack, but there doesn't seem to be anything there on it.

—

Yes, there actually is a MacsBug manual. I'm looking at the manual for MacsBug 6.1 right now, and on pages 107-109 are brief examples of how to use DebugStr in C, Pascal, and Assembly. MacsBug 6.2 hasn't really changed any of this since 6.1 (to my knowledge), but the MacsBug 6.2 manual might be a little more verbose if you can wait a little while to get it.

MacsBug 6.2 is final, so the manual shouldn't be too far behind. You can purchase it from APDA. They can be reached at 800-282-2732.

As for other documentation on this, the only other thing I could find is a little piece of sample code using DebugStr in the Tech Note "Bugs In MacApp? Yes, But I Love It!," and a brief mention of how to properly exit debuggers in *Inside Macintosh* Volume VI on the System 7.0b4 CD.

Disable Virtual Memory when using MacsBug Step Spy
Date Written: 4/3/92
Last reviewed: 5/21/92

Step Spy seems to work less reliably with later versions of MacsBug. I get what acts like a double bus fault a lot—that is, the interrupt key is as effective as pressing the Apple logo on the front. I like the feature and would like to see it supported in the future. Any feedback?

—

MacsBug Step Spy command does not currently work with Virtual Memory enabled. Please avoid using it with VM, at least until such time as the release notes for a future MacsBug state that this has been corrected.

Building MacsBug dcmds
Date Written: 2/25/92
Last reviewed: 5/21/92

I'm having trouble building the dcmd samples on the E.T.O. CD-ROM. What am I doing wrong?

—

There are a number of problems with the build instructions for dcmd samples, mostly stemming from the fact that they have never been updated for MPW 3.2.

There are several things to note:

When you compile Printf.c, you may get a warning about p2cstr() being a function with no prototype. You can eliminate that by adding a "#include <Strings.h>" statement.

DRuntime.o is no longer needed. It was basically a copy of the old CRuntime.o, which like CInterface.o is not needed with MPW 3.2.

For BuildDcmd to work correctly, all code must be in segment Main. This requires several more "-sg" options in the Link command line. If BuildDcmd complains about too many segments, use the -map Link option to see which segments need to be remapped into segment Main.

I have put together a Makefile that solves these problems. It is included below. It is called Printf.Make and should be placed in the folder with Printf.c. It assumes all the files it needs are in the same relative folder positions as they are on ETO.

```
#   File:       printf.make
#   Target:     printf
```

```
#   Sources:    Printf.c
#   Created:    Wednesday, January 22, 1992 4:17:33 PM

#    dcmdGlue.a.o must be first in the object list
OBJECTS = ∂
       '::dcmd Libraries:dcmdGlue.a.o' ∂
       '::dcmd Libraries:put.c.o' ∂
       Printf.c.o
```

```
printf ƒƒ printf.make {OBJECTS}
   Link -msg nodup -t APPL -c '????' ∂
       -sg Main=STDIO -sg Main=SANELIB -sg Main=STDCLIB∂
       {OBJECTS} ∂
       #"{CLibraries}"CSANELib.o ∂
       #"{CLibraries}"Math.o ∂
       #"{CLibraries}"Complex.o ∂
       "{CLibraries}"StdClib.o ∂
       "{Libraries}"Runtime.o ∂
       "{Libraries}"Interface.o ∂
       -o printf
   ::BuildDcmd Printf 1004
   # If you want the dcmd to be loaded directly into Debugger Prefs,
   #   uncomment the following line
   #Echo 'include "Printf";'    |    Rez -a -o "{systemFolder}Debugger Prefs"

Printf.c.o ƒ printf.make Printf.c
    C -r  Printf.c -i '::dcmd Includes:'
```